图文 97 NameServer是如何初始化基于Netty的网络通信架构的?

详情 评论

NameServer是如何初始化基于Netty的网络通信架构的?

继《从零开始带你成为JVM实战高手》后，阿里资深技术专家携新作再度出山，重磅推荐：

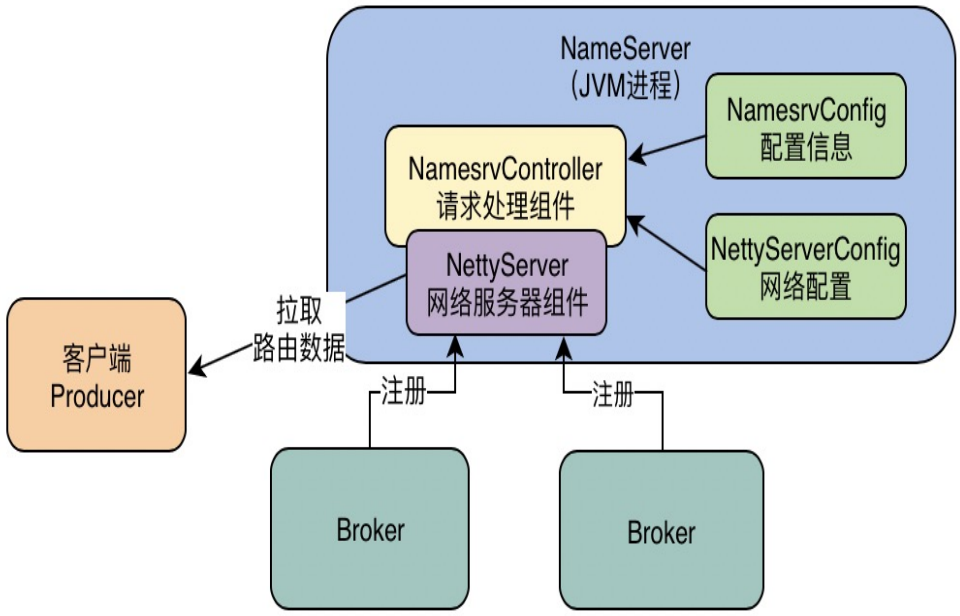
（点击下方蓝字试听）

《从零开始带你成为MySQL实战优化高手》

1、简单回顾：NamesrvController是如何被创建出来的?

我们先来简单的回顾一下，NamesrvController这个非常核心的组件是个什么东西以及是如何被创建出来的?

简单来说，大家先看一眼下面的图，是上一讲我们画出来的一个初步的NameServer架构图。



我们上一篇文章分析到，NameServer启动的时候，实际上会解析配置文件，然后初始化NamesrvConfig和NettyServerConfig两个核心配置类

然后我们进一步知道NamesrvConfig里其实没什么关键的东西，最主要的还是NettyServerConfig里包含的一些网络通信的参数，他们基本都有自己的默认值。

然后基于这两个核心配置类，实际上最后我们在源码里看到，他初步的构建出来了NamesrvController这个核心组件，如下面的代码片段，在createNamesrvController()这个方法中，最终是创建出了这个核心组件的。

```
public static NamesrvController createNamesrvController(String[] args)
        throws IOException, JoranException {
    final NamesrvController controller = new NamesrvController(
            namesrvConfig, nettyServerConfig);
}
```

而且我们还初步推测了一下，看到Netty相关的字眼，那么NamesrvController内部肯定包含基于Netty实现的网络通信组件了

所以大家在上面的图里可以看到，我们画了一个Netty网络服务器，负责监听和处理Broker以及客户端发送过来的网络请求。

2、NamesrvController被创建出来了，Netty服务器就能启动?

我们都知道，你这个NamesrvController被创建之后，我们最关心的其实就是他里面的Netty服务器得启动，这样NameServer才能在默认的9876这个端口上接收Broker和客户端的网络请求，比如Broker注册自己，客户端拉取Broker路由数据，等等。

那你觉得NamesrvController被创建出来了，然后就万事大吉了?

当然不可能那么简单的，我们看下面的NamesrvController的构造函数，他里面其实就是保存了一些实例变量的值而已，根本没干什么实质性的事儿。

```
public NamesrvController(
        NamesrvConfig namesrvConfig,
        NettyServerConfig nettyServerConfig) {
  this.namesrvConfig = namesrvConfig;
  this.nettyServerConfig = nettyServerConfig;
  this.kvConfigManager = new KVConfigManager(this);
  this.routeInfoManager = new RouteInfoManager();
  this.brokerHousekeepingService = new BrokerHousekeepingService(this);
  this.configuration = new Configuration(
    log,
    this.namesrvConfig, this.nettyServerConfig
  );
  this.configuration.setStorePathFromConfig(
        this.namesrvConfig, "configStorePath");
}
```

上面构造函数的代码你看一眼就行了，其实我根本都懒得去解释，因为大家看到这里基本也就只能知道他里面给一堆实例变量赋值了，构造了一些对象。但是比如KVConfigManager、RouteInfoManager、BrokerHousekeepingService这些东西是什么，你现在也看不出来，你暂时也不需要知道。

但是我们有一点是可以肯定的，仅仅创建出来一个NamesrvController，那绝对是不够的，肯定后续还有一些关键的代码，必须要启动他里面的Netty服务器才是核心的工作，这样他才能接受网络请求！

3、NamesrvController是如何被启动的？

因此我们回到NameServer启动时候最核心的main0()方法里去，看看你构造了NamesrvController之后，接着应该是要干什么事情。

```
public static NamesrvController main0(String[] args) {
  try {
    NamesrvController controller = createNamesrvController(args);
    start(controller);
    // 省略下面的一些无关紧要的代码
  } catch (Throwable e) {
    e.printStackTrace();
    System.exit(-1);
  }
  return null;
}
```

在上面的代码里，我省略了下面一些无关紧要的代码，其实大家会发现，启动NameServer的逻辑是最清晰明了的，因为看下来他就两个步骤：

一个是我们之前分析过的创建NamesrvController，这个过程中会有一些解析配置文件的工作，之前讲过了。

另外下面就是最关键的一个步骤，就是start(controller)这个代码，他就是启动了NamesrvController这个核心的组件！

4、NamesrvController在启动时会干什么？

接着我们来看start(controller)这个方法里的代码逻辑，看看下面，我省略了一些代码，就看下start()方法开头的一些源码片段。

```
public static NamesrvController start(
        final NamesrvController controller) throws Exception {
  if (null == controller) {
    throw new IllegalArgumentException("NamesrvController is null");
  }
  boolean initResult = controller.initialize();
  if (!initResult) {
     controller.shutdown();
     System.exit(-3);
  }
  // 省略了下面的一些代码
}
```

在上面我们可以看到，最为关键的一行代码就是boolean initResult = controller.initialize()这个地方，他其实就是对NamesrvController执行了initialize初始化的操作。

既然是初始化，那么我们可以大胆的推测下，NamesrvController里我们最为关注的，不就是Netty服务器么，那么这个初始化的地方，是不是就是把他内部的Netty服务器给初始化构造出来了呢?

5、Netty服务器是如何初始化的?

进入到controller.initilize()方法内部，我省略了一些代码，就看开始进入的代码片段:

```
public boolean initialize() {
  this.kvConfigManager.load();
  this.remotingServer = new NettyRemotingServer(
        this.nettyServerConfig, this.brokerHousekeepingService);
}
```

看到这里，大家是不是就恍然大悟了，啥都不用多说了，刚开始有一个kvConfigManager.load()，我们大致推测可能就是在里面有一些kv配置数据，是这个组件管理的，然后这里可能就是从磁盘上加载了kv配置吧。

但是我们一定不要陷入一些无关紧要的源码流程里去，因为我们现在分析NameServer启动的源码，不就是想知道他是如何初始化网络通信架构的么？我们就想知道后续Broker和客户端给他发送请求的时候，他是怎么处理的!

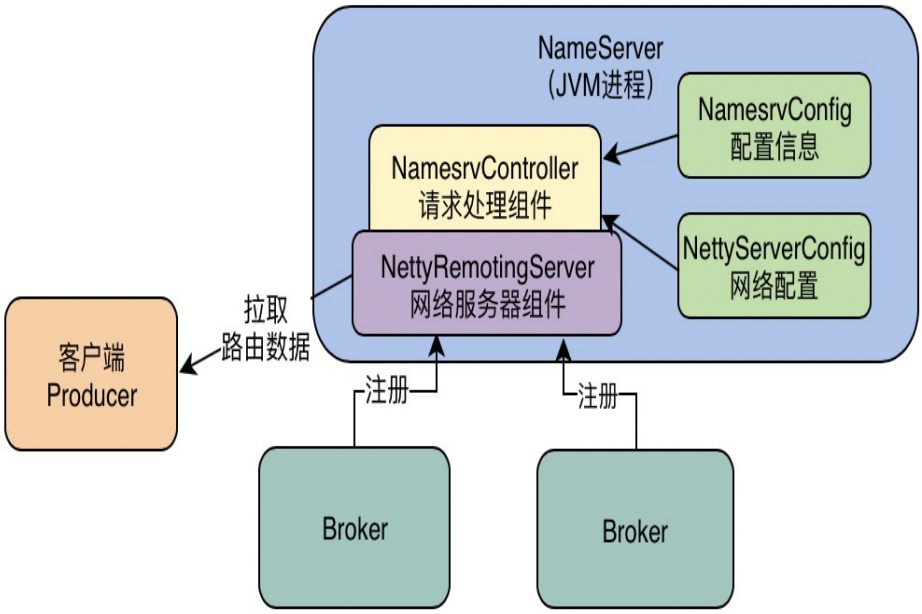
所以此时你只要对kvConfigManager有个印象就可以了，然后可以把他抛之脑后了，我们接着看下一行代码

this.remotingServer = new NettyRemotingServer(this.nettyServerConfig, this.brokerHousekeepingService)

这个就非常不得了了，很明显，他就是构造了一个NettyRemotingServer，也就是Netty网络服务器。

其实我们一直在找的就是这块代码，大家看到这里，先别太激动，也别太着急往下看，我们捋一捋思路，再看一眼下面那幅图，我把里面的NettyRemotingServer给写进去了，大家会发现核心的网络通信组件已经出来了。

在NamesrvController组件被构造好之后，接着进行初始化的时候，首先就是把核心的NettyRemotingServer网络服务器组件给构造了出来。



6、NettyRemotingServer是如何初始化的?

接着我们当然是应该看看NettyRemotingServer中的Netty服务器是如何初始化的了，这就得看看他的构造函数了

但是他的构造函数里代码很多，其实我就截取最为关键的一行代码给大家看就可以了。

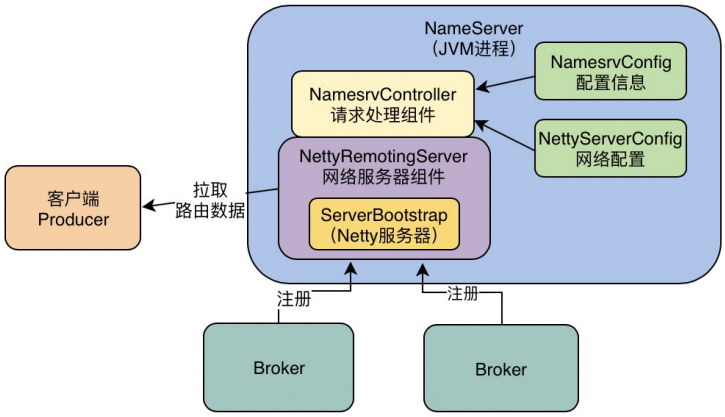
```
public NettyRemotingServer(
        final NettyServerConfig nettyServerConfig,
        final ChannelEventListener channelEventListener) {
    super(
        nettyServerConfig.getServerOnewaySemaphoreValue(),
        nettyServerConfig.getServerAsyncSemaphoreValue()
    );
    this.serverBootstrap = new ServerBootstrap();
    // 省略了一大堆的代码
}
```

上面的代码中，其实最为关键的一行代码就是：

this.serverBootstrap = new ServerBootstrap()

有的朋友可能没学习过Netty，不过不要紧，我直接告诉你就行了，这个ServerBootstrap，就是Netty里的一个核心的类，他就是代表了一个Netty网络服务器，通过这个东西，最终可以让Netty监听一个端口号上的网络请求。

讲到这里，我在图里又加入了一点东西，NettyRemotingServer是一个RocketMQ自己开发的网络服务器组件，但是其实底层就是基于Netty的原始API实现的一个ServerBootstrap，是用作真正的网络服务器的。



7、今日作业

相信大家一边跟着我的文章看一些核心源码片段，同时看一下我对源码的一些用最通俗的语言分析的思路，同时再加上我画的一些图，应该能够很容易理解RocketMQ的核心源码逻辑思路，同时结合图形可以快速的对源码执行过程中对应的架构原理有一个深刻的印象，最终你源码分析文章看完了，应该脑子里记住的就是我画的这些图。

同时只要你看懂了源码分析的文章，再自己认真的去完成我布置的一些源码分析的作业，自己复习和巩固源码思路，自己对照画好的图去会议源码执行流程和思路，那么你一定可以对RocketMQ的源码有一个较为深刻的理解的。

今日布置给大家的作业，就是结合我讲解的源码分析思路，自己去看一下NamesrvStratup中的main0()方法中的start(controller)这块启动NamesrvController的初步的一些源码，找其中关键的地方仔细看一下，理解NettyRemotingServer这个网络通信组件初始化的过程。

End
